



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/804,346	03/12/2001	Kenneth A. Chupa	CA920010012US1	1135

45541 7590 06/28/2007
HOFFMAN WARNICK & DALESSANDRO LLC
75 STATE ST
14TH FLOOR
ALBANY, NY 12207

EXAMINER

YIGDALL, MICHAEL J

ART UNIT	PAPER NUMBER
----------	--------------

2192

MAIL DATE	DELIVERY MODE
-----------	---------------

06/28/2007

PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary

Application No.

09/804,346

Applicant(s)

CHUPA ET AL.

Examiner

Michael J. Yigdall

Art Unit

2192

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 10 May 2007.
- 2a) ☐ This action is FINAL. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1-22 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1-22 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
2. ☐ Certified copies of the priority documents have been received in Application No. _____.
3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) ☐ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO/SB/08)
Paper No(s)/Mail Date _____
- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____
- 5) ☐ Notice of Informal Patent Application
- 6) ☐ Other: _____

DETAILED ACTION

1. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on February 28, 2007 has been entered. Claims 1-22 are pending.

Response to Arguments

2. Applicant's arguments have been fully considered but they are not persuasive.

Applicant contends, with regard to independent claims 1, 6, 11, 14, 15 and 18, that the cited references fail to disclose that each physical code generator generates physical code corresponding to one of a plurality of performed operations in an application (remarks, page 11, first paragraph). Applicant contends that instead, the physical code generators of the cited references process entire applications and do not independently process separate portions of a common application (remarks, page 11, first paragraph).

However, the examiner respectfully submits that Applicant's reasoning is not commensurate with the scope of the claims. Even if a physical code generator were to process an entire application, as per Applicant's characterization of the references, this does not preclude the physical code generator from corresponding to one of a plurality of performed operations in the application. The claims indicate only that the application (or the "input data" or the "single input") comprises a plurality of performed operations; the claims do not positively recite that a

plurality of physical code generators is necessary to process a common application wherein each physical code generator independently processes a separate portion of the application.

In the Goldberg reference, for example, each physical code generator such as the “JDBCJavaQueryObjectImplGenerator” generator (block 806 in FIG. 8) generates physical code corresponding to a query object (see, for example, column 12, lines 10-13) that performs operations for interacting with a database management system (see, for example, column 5, lines 44-64). The database management operations are merely one of a plurality of operations that the application performs. For example, the application further comprises business objects that perform operations for interacting with the query objects (see, for example, FIG. 2 and column 5, lines 26-31). Thus, Goldberg teaches an application that performs a plurality of operations, and teaches physical code generators that generate physical code corresponding to one of the plurality of performed operations.

Moreover, the Pepin reference teaches a plurality of code generators (see, for example, code generators 930, 980 in FIG. 7), wherein each code generator generates code corresponding to a component (see, for example, column 7, lines 51-65). Pepin discloses, “The component implements controls, forms and functionality associated with a developer’s created application” (column 2, lines 4-5), and further discloses that the application comprises a plurality of components “such as buttons, menus, or other user interface elements” (column 1, lines 17-20). In other words, Pepin teaches an application comprising a plurality of components, wherein each component performs a different operation, and teaches a plurality of code generators, wherein each code generator independently generates code for one of the components.

Applicant contends, with regard to independent claims 1, 11 and 14, that the cited references fail to teach or suggest that the application that is produced by the physical code generators is adapted for a particular target environment (remarks, page 11, second paragraph). Applicant states that in contrast, the physical code generators of the cited references each correspond to different environments, and states, "For example, one code module may be used to produce an entire application in a Java environment while another is used to produce an entire application in a Linux environment" (remarks, page 11, second paragraph).

However, while Applicant does not identify a particular reference, Applicant's example refutes Applicant's argument. An "entire application in a Java environment" is, in fact, an application adapted for a particular target environment. Likewise, an "entire application in a Linux environment" is an application adapted for a particular target environment. Thus, the application that is produced by the physical code generators is adapted for a particular target environment. The examiner respectfully submits that the language of the claims does not patentably distinguish them over the references. The examiner notes further that independent claims 6, 15 and 18 do not include the language pertinent to Applicant's argument.

Applicant contends, with regard to independent claims 1, 11, 14 and 15, and dependent claim 7, that the cited references fail to teach that the generator dictionary is adapted to be amended by the user to remove at least one code generator and replace the at least one code generator with a replacement code generator (remarks, pages 11-12).

However, the examiner respectfully reminds Applicant that one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. Furthermore, the test for obviousness is not that the claimed

Art Unit: 2192

invention must be expressly suggested in any one or all of the references. Rather, the test is what the combined teachings of the references would have suggested to those of ordinary skill in the art. See *In re Keller*, 642 F.2d 413, 208 USPQ 871 (CCPA 1981), and *In re Merck & Co.*, 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986).

First, as set forth in the Office action, Goldberg teaches that the generator dictionary is adapted to be amended by the user to add a new code generator (see, for example, column 13, lines 21-26). This suggests to those of ordinary skill in the art that similarly, the generator dictionary is adapted to be amended by the user to replace a code generator and with a replacement code generator.

Second, as Applicant acknowledges (remarks, page 12, top), Pepin teaches a generator dictionary (metadata 330/430) that is adapted to be amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3). In Pepin, replacing the “associated designer” (in metadata 330/430) with a replacement designer is analogous to replacing the “associated code generator” (also in metadata 330/430) with a replacement code generator. At any rate, the teachings of Pepin further suggest to those of ordinary skill in the art that the generator dictionary of Goldberg is adapted to be amended to remove at least one code generator and replace the at least one code generator with a replacement code generator.

Furthermore, while Applicant contends that the amendments in Pepin are made by the component supplier and not by the user (remarks, page 12, top), Brassard teaches instructions and options for a code generator that are adapted to be amended by a developer or user (see, for example, column 6, lines 36-46), as set forth in the Office action.

In response to Applicant's argument that "the Office's rejection of independent claim 1 seeks to combine four different references to support its obviousness rejection" and that "it is unfathomable how the Office can maintain its contention that there is motivation or suggestion in the references themselves or elsewhere to combine such diverse references" (remarks, page 12, last paragraph), the examiner respectfully reminds Applicant that reliance on a large number of references in a rejection does not, without more, weigh against the obviousness of the claimed invention. See *In re Gorman*, 933 F.2d 982, 18 USPQ2d 1885 (Fed. Cir. 1991).

Reasons for combining the references were provided in the Office action. That the combinations are somehow unfathomable to Applicant is not evidence that they are "flawed" (remarks, page 13, top).

Claim Rejections - 35 USC § 101

3. 35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

4. Claims 1-5 and 15-17 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

With respect to claim 1 (currently amended), the claim is directed to a "computer system for generating source code." However, as recited, the computer system is reasonably interpreted as entirely software. There are no hardware components or computer-readable media included in the claim. Accordingly, the recited computer system amounts to descriptive material *per se*,

Art Unit: 2192

which does not fall within any category of statutory subject matter. See MPEP § 2106.01.

Dependent claims 2-5 (original) do not remedy claim 1.

With respect to claim 15 (currently amended), the claim is directed to a “code generation framework tool.” However, as recited, the code generation framework tool is reasonably interpreted as entirely software. There are no hardware components or computer-readable media included in the claim. Accordingly, the recited code generation framework tool amounts to descriptive material *per se*, which does not fall within any category of statutory subject matter. See MPEP § 2106.01. Dependent claims 16 (original) and 17 (previously presented) do not remedy claim 15.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

6. Claims 1-5 and 11-22 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 6,496,833 to Goldberg et al. (art of record, “Goldberg”) in view of U.S. Patent No. 6,948,150 to Pepin (art of record, “Pepin”) in view of U.S. Patent No. 6,738,967 to Radigan (art of record, “Radigan”) and in view of U.S. Patent No. 6,742,175 to Brassard (art of record, “Brassard”).

With respect to claim 1 (currently amended), Goldberg discloses a computer system for generating source code (see, for example, the abstract), said computer system comprising:

(a) a user amendable generator dictionary having a plurality of entries associating a generator routine with a generator identity, said generator identity identifying a code generator (see, for example, FIG. 8 and column 12, lines 32-50, which shows a programming construct or dictionary with a plurality of entries that associates a generator routine for a specific language and database with a generator class name or identity that identifies a code generator, and column 13, lines 21-26, which shows that the dictionary is amendable) and said generator dictionary comprising at least one logical generator and at least one physical code generator (see, for example, column 12, lines 18-27, which shows the QueryObjectImplGenerator base class, a logical generator, and implementation classes, i.e. physical code generators).

Goldberg further discloses that each physical code generator generates physical code corresponding to one of a plurality of performed operations of an application (see, for example, column 12, lines 10-13, which shows that the code generator generates physical code corresponding to a query object, and column 5, lines 44-64, which further shows that the query object performs operations such as interacting with a DBMS) adapted for a particular target environment (see, for example, column 13, lines 16-18, which shows a particular target environment comprising Java and JDBC database drivers).

Furthermore, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator generates code corresponding to a component (see, for example, column 7, lines 51-65) that performs one of a

Art Unit: 2192

plurality of operations of an application (see, for example, column 1, lines 17-20 and column 2, lines 4-7), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that each physical code generator generates physical code corresponding to one of a plurality of performed operations of an application adapted for a particular target environment, so that the implementation of each performed operation can be changed independently of and without modifying the generator tool itself.

Goldberg does not expressly disclose that each of the at least one physical code generators is independent from each other of the at least one physical code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that each of the at least one physical code generators is independent from each other of the at least one physical code generators, so that the code generators can be independently upgraded.

Although Goldberg does not expressly disclose the limitation wherein the generator dictionary is adapted to be amended by the user to remove at least one code generator and to replace the at least one code generator with a replacement code generator, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

Goldberg further discloses:

Art Unit: 2192

(b) a code generation framework tool wherein said code generation framework tool, responsive to a request for an invocation of said generator routine, invokes said code generator identified by said generator identity associated with said generator routine (see, for example, column 11, lines 21-26, which shows a generator tool responding to input, i.e. to a request, and invoking the identified code generator);

wherein the at least one logical generator calls the at least one physical code generator to generate source code (see, for example, FIG. 8 and column 12, line 62 to column 13, line 20, which shows a logical generator such as `JavaQueryObjectImplGenerator` deriving or calling a physical code generator such as `JDBCJavaQueryObjectImplGenerator` to generate source code).

With respect to claim 2 (original), Goldberg further discloses the limitation wherein said generator dictionary comprises a plurality of generator routines, each of said generator routines associated with a generator identity (see, for example, FIG. 8 and column 12, lines 32-50, which shows a plurality of generator routines associated with a generator identity).

With respect to claim 3 (original), Goldberg further discloses the limitation wherein said generator dictionary comprises a text file (see, for example, column 12, lines 32-50, which shows that the dictionary is in the form of source code, inherently comprising a text file).

With respect to claim 4 (original), Goldberg further discloses the limitation wherein said generator routine comprises a logical generator name (see, for example, column 12, lines 32-50, which shows that the generator routine comprises a logical generator name such as “`sybase_ctlib`” or “`oracle_oci`”).

With respect to claim 5 (original), Goldberg further discloses the limitation wherein said code generation framework tool retrieves from said generator dictionary said generator identity responsive to said request (see column 11, lines 21-26, which shows that the generator tool selects the appropriate generator in response to the request).

With respect to claim 11 (currently amended), Goldberg discloses a method of generating source code for a first and a second deployment environment from a single input (see, for example, FIG. 8 and column 12, lines 9-51, which shows code generators for generating source code for a plurality of platforms or deployment environments), said method comprising:

(a) invoking a first code generator to generate source code for said first deployment environment from said single input, said first code generator identified by retrieving code generator identity data from a user amendable generator dictionary based on a generator routine (see, for example, column 11, lines 21-26, which shows invoking the appropriate code generator based on input information, and column 12, lines 18-50, which shows retrieving the identity of a code generator from a generator dictionary based on a generator routine for the deployment environment, i.e. the first deployment environment, and column 13, lines 21-26, which shows that the dictionary is amendable).

Goldberg further discloses that the first code generator generates source code and corresponds to one of a plurality of performed operations of the single input in the first deployment environment (see, for example, column 12, lines 10-13, which shows that the code generator generates source code corresponding to a query object, and column 5, lines 44-64, which further shows that the query object performs operations such as interacting with a DBMS).

Furthermore, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator generates code corresponding to a component (see, for example, column 7, lines 51-65) that performs one of a plurality of operations of an application (see, for example, column 1, lines 17-20 and column 2, lines 4-7), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that the first code generator generates source code corresponding to a one of a plurality of performed operations of the single input in the first deployment environment, so that the implementation of each performed operation can be changed independently of and without modifying the generator tool itself.

Goldberg does not expressly disclose that the source code is generated independently of each other code generator.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the source code is generated independently of each other code generator, so that the code generators can be independently upgraded.

(b) modifying said generator dictionary to associate a second code generator with said generator routine (see, for example, column 13, lines 21-26, which shows adding new code generator classes, i.e. modifying the generator dictionary, to associate other code generators).

Although Goldberg does not expressly disclose removing the first code generator and replacing the first code generator the second code generator, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

(c) invoking said second code generator to generate source code for said second deployment environment from said single input, said second code generator identified by retrieving code generator identity data from said generator dictionary based on said generator routine (see, for example, column 11, lines 21-26, which shows invoking the appropriate code generator based on input information, and column 12, lines 18-50, which shows retrieving the identity of a code generator from the generator dictionary based on a generator routine for the deployment environment, i.e. the second deployment environment), wherein the second code generator generates source code corresponding to the performed operation of the single input in the second deployment environment (see the explanation above).

With respect to claim 12 (original), Goldberg further discloses the limitation wherein said invoking said first code generator comprises a call issued by one of a code generation framework tool and a code generator; and wherein said invoking said first code generator comprises a call issued by one of said code generation framework tool and a code generator (see, for example, column 11, lines 21-26, which shows invoking or calling the appropriate code generator).

With respect to claim 13 (original), Goldberg further discloses the limitation wherein said modifying comprises editing said generator dictionary (note that modifying the generator dictionary inherently comprises editing the generator dictionary).

With respect to claim 14 (currently amended), Goldberg discloses a generator dictionary stored on a recordable medium comprising:

a plurality of generator routines that include at least one logical generator and at least one physical code generator (see, for example, column 12, lines 18-27, which shows the QueryObjectImplGenerator base class, a logical generator, and implementation classes, i.e. physical code generators), each of said generator routines associated with code generator identity data (see, for example, FIG. 8 and column 12, lines 32-50, which shows a programming construct or dictionary comprising a plurality of generator routines for specific languages and databases associated with code generator class names or identities).

Goldberg further discloses that each of the plurality of generator routines generates source code corresponding to one of a plurality of performed operations in an application (see, for example, column 12, lines 10-13, which shows that the code generator generates source code corresponding to a query object, and column 5, lines 44-64, which further shows that the query object performs operations such as interacting with a DBMS) adapted for a particular target environment (see, for example, column 13, lines 16-18, which shows a particular target environment of a code generator comprising Java and JDBC database drivers).

Furthermore, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator generates code corresponding to a component (see, for example, column 7, lines 51-65) that performs one of a plurality of operations of an application (see, for example, column 1, lines 17-20 and column 2,

Art Unit: 2192

lines 4-7), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that each of the plurality of generator routines generates source code corresponding to one of a plurality of performed operations in an application adapted for a particular target environment, so that the implementation of each performed operation can be changed independently of and without modifying the generator tool itself.

Goldberg does not expressly disclose that the at least one physical code generator is independent from each other of the at least one physical code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the at least one physical code generator is independent from each other of the at least one physical code generators, so that the code generators can be independently upgraded.

wherein the at least one logical generator calls the at least one physical code generator to generate source code (see, for example, FIG. 8 and column 12, line 62 to column 13, line 20,

Art Unit: 2192

which shows a logical generator such as `JavaQueryObjectImplGenerator` deriving or calling a physical code generator such as `JDBCJavaQueryObjectImplGenerator` to generate source code), and

wherein the generator dictionary is designed to be amended by a user (see, for example, column 13, lines 21-26, which shows that the dictionary is amendable).

Although Goldberg does not expressly disclose removing at least one of the plurality of generator routines and replacing the at least one of the plurality of generator routines with a replacement generator routine, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

With respect to claim 15 (currently amended), Goldberg discloses a code generation framework tool (see, for example, the abstract) comprising:

(a) a receiver for receiving input data (see, for example, GUI 634 in FIG. 6 and column 10, lines 47-53, which shows an interface or receiver for receiving input data);

(b) a user amendable generator dictionary accessor for retrieving data from a generator dictionary comprising at least one logical generator and at least one physical code generator (see, for example, column 12, lines 29-51, which shows an accessor method for retrieving data from a generator dictionary, and column 12, lines 18-27, which shows the QueryObjectImplGenerator base class, a logical generator, and implementation classes, i.e. physical code generators, and column 13, lines 21-26, which shows that the dictionary is amendable).

Goldberg does not expressly disclose that the at least one physical code generator is independent from each other of the at least one physical code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the at least one physical code generator is independent from each other of the at least one physical code generators, so that the code generators can be independently upgraded.

(c) an invoking mechanism for calling a code generator (see, for example, code generator 604 in FIG. 6 and column 11, lines 21-26, which shows invoking a code generator); and

wherein, responsive to a receipt of input data at said receiving, said invoking mechanism calls a code generator identified by identity data retrieved by said generator dictionary accessor from a generator dictionary (see, for example, column 11, lines 21-26, which shows invoking a code generator in response to input data, and column 12, lines 18-51, which shows that the identity of a code generator is retrieved from the accessor method).

Goldberg further discloses that the identified code generator corresponds to one of a plurality of performed operations of the input data (see, for example, column 12, lines 10-13, which shows that the code generator corresponds to a query object, and column 5, lines 44-64, which further shows that the query object performs operations such as interacting with a DBMS).

Furthermore, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator corresponds to a component (see, for example, column 7, lines 51-65) that performs one of a plurality of operations of an application (see, for example, column 1, lines 17-20 and column 2, lines 4-7), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that the identified code generator corresponds to one of a plurality of performed operation of the input data, so that the implementation of each performed operation can be changed independently of and without modifying the generator tool itself.

Although Goldberg does not expressly disclose the limitation wherein the generator dictionary is adapted to be amended by the user to remove the code generator and replace the code generator with a replacement code generator, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

With respect to claim 16 (original), Goldberg further discloses a data dictionary associating a generator routine with identity data identifying a code generator (see, for example, column 12, lines 9-51, which shows a programming construct or dictionary that associates a generator routine for a specific language and database with a generator class name or identity, which identifies a code generator).

With respect to claim 17 (previously presented), Goldberg further discloses the limitation wherein said generator dictionary accessor identifies a generator routine within said input data received and wherein said code generator identified is determined by retrieving said identity data associated with said generator routine identified (see, for example, column 12, lines 29-51, which shows the accessor method for identifying a code generator based on a generator routine, which is determined from the input data specifying a language and database).

With respect to claim 18 (currently amended), Goldberg discloses a computer readable medium storing instructions and data (see, for example, column 20, lines 5-11), said instructions and data for adapting a computer system to:

(a) responsive to a request for invoking a generator routine, identify, in a user amendable generator dictionary that includes at least one logical generator and at least one physical code generator, a code generator associated with said generator routine (see, for example, column 11,

Art Unit: 2192

lines 21-26, which shows selecting the appropriate generator in response to a user request, and column 12, lines 18-50, which shows identifying the code generator from a generator dictionary, including the QueryObjectImplGenerator base class, a logical generator, and implementation classes, i.e. physical code generators, and column 13, lines 21-26, which shows that the dictionary is amendable).

Goldberg further discloses that the identified code generator corresponds to one of a plurality of performed operations of the input data (see, for example, column 12, lines 10-13, which shows that the code generator corresponds to a query object, and column 5, lines 44-64, which further shows that the query object performs operations such as interacting with a DBMS).

Furthermore, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator corresponds to a component (see, for example, column 7, lines 51-65) that performs one of a plurality of operations of an application (see, for example, column 1, lines 17-20 and column 2, lines 4-7), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that the identified code generator corresponds to one of a plurality of performed operations of the input data, so that the implementation of each performed operation can be changed independently of and without modifying the generator tool itself.

Goldberg does not expressly disclose that the at least one physical code generator is independent from each other of the at least one physical code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the at least one physical code generator is independent from each other of the at least one physical code generators, so that the code generators can be independently upgraded.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

Art Unit: 2192

(b) pass said input data to said code generator identified (see, for example, column 11, lines 7-20, which shows input data specifying the operating environment, and column 12, lines 18-27, which shows passing the data to a generator method), said code generator being operable to:

call another code generator to generate the source code (see, for example, column 12, line 62 to column 13, line 20, which shows a generator such as `JavaQueryObjectImplGenerator` deriving or calling another generator such as `JDBCJavaQueryObjectImplGenerator` to generate source code); and

generate the source code (see, for example, column 11, lines 21-26, which shows generating source code).

With respect to claim 19 (previously presented), the limitations recited in the claim are analogous to those of claim 7 (see the rejection of claim 7 below).

With respect to claim 20 (previously presented), the limitations recited in the claim are analogous to those of claim 8 (see the rejection of claim 8 below).

With respect to claim 21 (previously presented), the limitations recited in the claim are analogous to those of claim 9 (see the rejection of claim 9 below).

With respect to claim 22 (previously presented), the limitations recited in the claim are analogous to those of claim 10 (see the rejection of claim 10 below).

7. Claims 6 is rejected under 35 U.S.C. 103(a) as being unpatentable over Goldberg in view of Pepin and in view of Radigan.

With respect to claim 6 (currently amended), Goldberg discloses a method for generating source code from input data (see, for example, the abstract), said method comprising:

(a) responsive to a request for invoking a generator routine, identifying from a plurality of code generators a code generator associated with said generator routine (see, for example, column 11, lines 21-26, which shows selecting the appropriate generator in response to a user request).

Goldberg further discloses that the identified code generator corresponds to one of a plurality of performed operations of the input data (see, for example, column 12, lines 10-13, which shows that the code generator corresponds to a query object, and column 5, lines 44-64, which further shows that the query object performs operations such as interacting with a DBMS).

Furthermore, Pepin, in an analogous art (see, for example, the abstract), discloses a generator dictionary that identifies a plurality of code generators (see, for example, metadata 900 and code generators 930, 980 in FIG. 7), wherein each code generator corresponds to a component (see, for example, column 7, lines 51-65) that performs one of a plurality of operations of an application (see, for example, column 1, lines 17-20 and column 2, lines 4-7), so that the implementation of the component can be changed independently of and without modifying the design tool itself (see, for example, column 2, lines 7-16).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Pepin, such that the identified code generator corresponds to one of a plurality of performed operations of the input data, so that the implementation of each performed operation can be changed independently of and without modifying the generator tool itself.

Art Unit: 2192

(b) passing said input data to said code generator identified (see, for example, column 11, lines 7-20, which shows input data specifying the operating environment, and column 12, lines 18-27, which shows passing the data to a generator method), said code generator being operable to:

call another code generator to generate the source code (see, for example, column 12, line 62 to column 13, line 20, which shows a generator such as `JavaQueryObjectImplGenerator` deriving or calling another generator such as `JDBCJavaQueryObjectImplGenerator` to generate source code); and

generate the source code (see, for example, column 11, lines 21-26, which shows generating source code).

Goldberg does not expressly disclose that the source code is generated independently of other code generators.

However, Radigan, in an analogous art (see, for example, the abstract), discloses a plurality of translators and optimizers that generate code for different processor architectures, wherein each translator and optimizer is independent from each other translator and optimizer (see, for example, column 9, lines 4-25). This allows each translator to be independently upgraded to support a new version of its respective processor architecture (see, for example, column 9, lines 26-32).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Radigan, such that the source code is generated independently of other code generators, so that the code generators can be independently upgraded.

8. Claims 7-10 are rejected under 35 U.S.C. 103(a) as being unpatentable over Goldberg in view of Pepin and in view of Radigan, as applied to claim 6 above, and further in view of Brassard.

With respect to claim 7 (previously presented), Goldberg further discloses the limitation wherein said identifying comprises retrieving from a user amendable generator dictionary code generator identity data associated with said generator routine (see, for example, FIG. 8 and column 12, lines 27-51, which shows a programming construct or dictionary for retrieving the identity of a code generator associated with a generator routine that represents a specific language and database, and column 13, lines 21-26, which shows that the dictionary is amendable).

Although Goldberg does not expressly disclose the limitation wherein the generator dictionary is adapted to be amended by the user to remove at least one of the plurality of code generators and to replace the at least one of the plurality of code generators with a replacement code generator, it would have been obvious to one of ordinary skill in the art at the time the invention was made that, like adding a new code generator to the generator dictionary of Goldberg (see, for example, column 13, lines 21-26), a user could also remove an existing code generator and replace it with a new code generator. Pepin, for example, discloses that the metadata or the generator dictionary can be amended to change or replace the implementation of a component (see, for example, column 6, lines 8-18). Specifically, Pepin illustrates an example in which the metadata 330/430 is amended to replace designer 340/440 with a replacement

Art Unit: 2192

designer 370/470 (see, for example, FIGS. 2 and 3), the effect of which is that designer 340/440 is removed from the metadata 330/430.

Although Goldberg is silent as to which user or users may amend the generator dictionary, Brassard, in an analogous art (see, for example, the abstract), discloses generator instructions and options that are amended by a developer (see, for example, column 6, lines 36-46), so that the developer can customize the code generator to suit the developer's needs (see, for example, column 12, lines 58-67).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to supplement Goldberg with the features taught by Brassard, such that the generator dictionary is adapted to be amended by a user such as a developer, so that the developer can customize the generator dictionary to suit the developer's needs.

With respect to claim 8 (original), Goldberg further discloses the limitation wherein said identifying further comprises prior to said retrieving, locating said generator routine in said generator dictionary (see, for example, column 12, lines 32-50, which shows a "switch" construct, which inherently involves locating the appropriate "case" statement before returning the identity of a code generator).

With respect to claim 9 (original), Goldberg further discloses the limitation wherein said generator dictionary comprises a lookup table (see, for example, column 12, lines 32-50, which shows a programming construct or dictionary that serves as a lookup table).

With respect to claim 10 (original), Goldberg further discloses the limitation wherein said generator dictionary comprises a text file (see, for example, column 12, lines 32-50, which shows that the dictionary is in the form of source code, inherently comprising a text file).

Conclusion

9. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Michael J. Yigdall whose telephone number is (571) 272-3707. The examiner can normally be reached on Monday through Friday from 7:30am to 4:00pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (571) 272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

MY

Michael J. Yigdall
Examiner
Art Unit 2192

mjy



**TUAN DAM
SUPERVISORY PATENT EXAMINER**